Leevi Pulkkinen, 910776, Energia- ja ympäristötekniikka, 5.5.2021

Yleiskuvaus:

Ohjelma on yksinkertainen tasohyppelypeli, jossa pelaaja ohjaa palloa aloituspisteestä maaliin ja samalla väistelee piikkejä ja kerää kolikoita. Ohjelman käynnistyessä luodaan pieni ikkuna, jonka sisällä on graafinen käyttöliittymä. Pelissä on kolme pelattavaa tasoa. Kun pelaaja saapuu maaliin, siirrytään automaattisesti seuraavalle tasolle. Kun pelaaja pääsee viimeisellä tasolla maaliin, pelaaja voittaa pelin. Kun peli on voitettu, näytölle ilmestyy voittonäkymä. Voittonäkymässä on pistetaulu ja kaksi nappia, joiden avulla voi pelata uudelleen tai lopettaa pelin. Pistetaulu lukee tiedostosta 5 parasta pistemäärää ja näyttää ne pistetaulukossa.

Projektisuunnitelmasta poiketen ohjelmassa ei ole aloitusnäyttöä, eikä liikkuvia vihollisia. Tämä johtuu osittain ajan puutteesta ja osittain siitä, että en kokenut näiden ominaisuuksien olevan ohjelman kokonaisuuden kannalta välttämättömän tärkeitä. Työ on toteutettu vaikeusasteella VAATIVA.

Käyttöohje:

Kun ohjelman kaikki tiedostot ovat ladattuina, voi ohjelman käynnistää ajamalla ”game.py” nimistä tiedostoa. Ohjelman käynnistyttyä voidaan nuolinäppäimillä ohjata ruudulla näkyvää punaista palloa. Pallo liikkuu sivuille painamalla nuolinäppäimiä oikealle tai vasemmalle ja pallo hyppää painamalla nuolinäppäintä ylöspäin. Peliä voi pelata niin kauan kuin haluaa, ja se on helppo aloittaa alusta voittamisen jälkeen. Pelin voi sulkea painamalla ikkunan oikeassa yläkulmassa olevaa ruksia tai käyttämällä pikanäppäintä ”ctrl + q”. Käynnissä olevan tason voi myös aloittaa alusta painamalla ”ctrl + r”.

Ulkoiset kirjastot:

Ohjelmassa on käytetty vain PyQt5 kirjastoa. PyQt5 kirjaston avulla luotiin graafinen käyttöliittymä ja luotiin myös suuri osa tasolla esiintyvistä objekteista hyödyntämällä perintää.

Ohjelman rakenne:

Scene luokka:

Scene on selkeästi ohjelman suurin luokka ja se toimii koko ohjelman pääluokkana. Scene luokka luo kaikkien muiden luokkien instanssit, pitää kirjaa mitä nappeja painetaan ja sisältää kaikki tärkeimmät metodit, jotka muokkaavat ikkunan näkymää. Scene luokka perii PyQt5 luokan QGraphicsScene. QGraphicsScenen periminen pääluokassa on mielestäni looginen ratkaisu, koska QGraphicsScene on alusta, jolle kaikki luodut QGraphicsItemit sijoitetaan. Tämän lisäksi monet muut luokat tarvitsevat tiedon siitä mikä on scene jonne Itemit tulisi lisätä. Tämä onnistuu helposti lisäämällä self (eli Scene) luotavien luokkien attribuutiksi.

Scene luokka sisältää tärkeitä metodeja kuten:

init_scene: Luo koko ohjelman perustan ja mahdollistaa myös helpon tavan pelata peliä uudestaan alkutilanteesta.

keyPressEvent ja keyReleaseEvent: Tarkistavat mitä nappeja painetaan.

timerEvent: Päivittää pelaajaa ja tasoa aina kun käynnistetty kello tikittää.

uusi_taso ja restart_level: Mahdollistavat uuden tason luomisen ja nykyisen tason aloittamisen alusta.

Scene luokassa luodaan myös ruudun vasemmassa yläkulmassa oleva pistelaskuri, ja pelin jälkeen esiintyvä pistetaulu.

Ikkuna luokka:

Ikkuna luokan instanssi luodaan Scene luokan init_scene metodissa. Ikkuna luokka luo ruudulle ikkunan, jossa kaikki ohjelman graafinen toiminta näytetään. Ikkuna luokan metodeilla exit_button, replay_button ja luo_pikanappaimet luodaan voittoruudussa näkyvät kaksi nappia, sekä pikanäppäimet, joita voi käyttää koko pelin ajan. Ikkunan toteuttaminen omassa luokassaan helpotti nappien, sekä pikanäppäinten luontia. Tämän lisäksi koodin rakenne on hieman siistimpi, koska ikkunaa ja sen metodeja ei tarvitse tunkea jo suureen Scene luokkaan.

Scene ja Ikkuna luokat luovat ohjelman perustan ja alustavat graafisen käyttöliittymän.

Taso luokka:

Taso luokka luo annetun datan pohjalta pelattavan tason ja piirtää sen luotuun ikkunaan. Taso luodaan Scene luokan alussa ja uusi taso luodaan aina kun pelaaja pääsee maaliin (paitsi viimeisen tason jälkeen).

Taso luokan metodi lue_data on yksi ohjelman tärkeimmistä. lue_data käy läpi annetun tason rakenteen ja tarkastelee kohta kerrallaan mitä sceneen tulisi lisätä. lue_data aloittaa tarkastelun ruudun vasemmasta yläkulmasta ja käyttää hyväkseen koordinaatteja, joiden avulla objektit luodaan oikeaan paikkaan. Kaikki tason Maa, Piikki, Maali ja Kolikko oliot luodaan lue_data metodissa. lue_data metodissa on myös tarkistus joka heittää poikkeuksen, jos tason datan rakenteessa on virhe jota ei voida ohittaa. Taso luokassa on myös kolme eri listaa: self.objects, self.viholliset ja self.kolikot. Näihin listoihin lisätään erilaiset tasolle luodut objektit. Erillisten listojen luonti erilaisille objekteille on kätevää, koska pelaajan törmääminen eri objekteihin aiheuttaa erilaisia toimintoja. Kun kaikki saman tyyppiset objektit ovat omassa listassaan, on helppoa käydä lista läpi ja tarkastella tapahtuuko törmäystä.

Maa, Piikki, Maali ja Kolikko luokat ovat yksinkertaisia ja sisältävät pääosin vain graafisen osuuden kyseisten luokkien toiminnasta. Tämä johtuu siitä, että näiden luokkien toiminta perustuu tömäykseen pelaajan kanssa ja törmäykset tarkastellaan erillisessä pelaaja luokassa. Näistä luokista luodaan todella monia instansseja, joten on silti tärkeää että ne ovat omat luokkansa, vaikka tämä ei olisikaan 100% välttämätöntä.

Maa luokka:

Maa luokka luo yksinkertaisen neliön tasolle, johon pelaaja voi törmätä. Maa luokka on todella lyhyt mutta mielestäni se oli kuitenkin koodin siisteyden takia hyvä pitää omana luokkanaan kuten muutkin tason objektit. Maa luokka perii QGraphicsRectItem luokan, jotta Maa olioita on helpompi käsitellä ja vertailla muihin objekteihin.

Piikki luokka:

Piikki luokka muistuttaa pitkälti Maa luokkaa. Suurin ero on, että piikki perii luokan QGraphicsPolygonItem, jotta piikeistä voidaan tehdä kolmion mallisia. Vaikka Piikki ja Maa luokka ovat melko samanlaisia niiden käyttötapa ohjelmassa on hyvin erilainen, mutta nämä toiminnot määritellään Pelaaja luokassa.

Maali luokka:

Maali luokassa luodaan nelikulmio, joka toimii porttina tasojen välillä. Jälkeenpäin ajateltuna Maalin ei välttämättä tarvitsisi olla oma luokkansa, koska maaleja on vain yksi jokaisella tasolla, mutta tuskin tästä haittaakaan on. Myös Maali luokka perii QGraphicsRectItemin

Kolikko luokka:

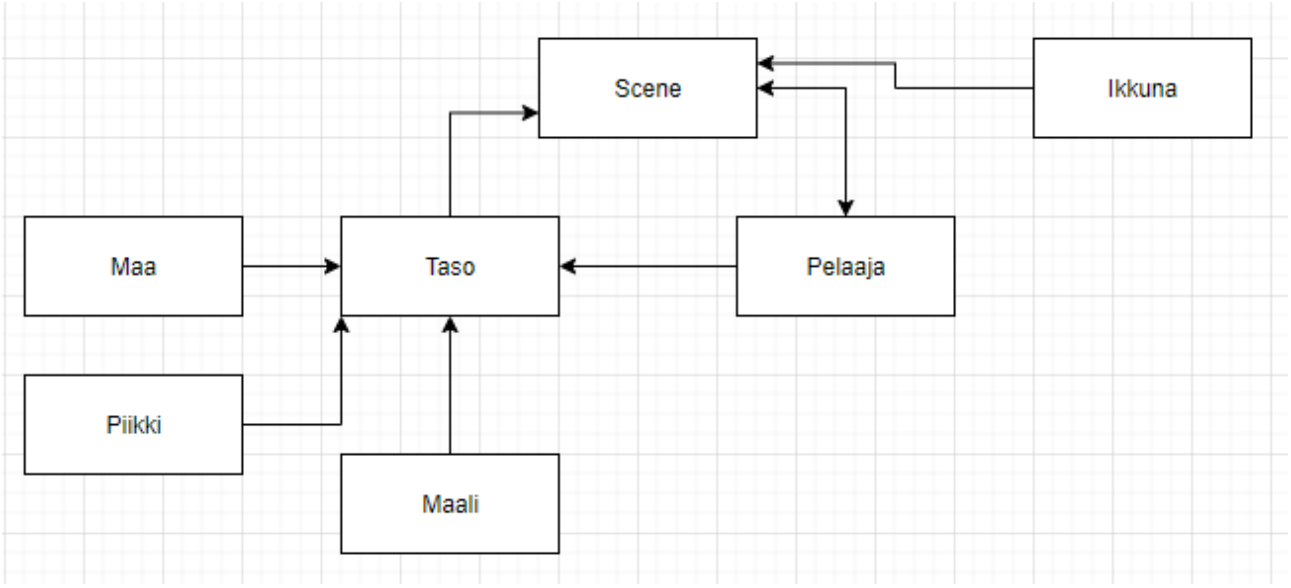
Kolikko luokassa luodaan tasolla näkyvät keltaiset kolikot, joita voi kerätä. Kolikko luokka perii QGraphicsEllipseItemin, koska kolikot ovat pyöreitä.

Taso, Maa, Piikki, Maali ja Kolikko hoitavat suuren osan ohjelman graafisen käyttöliittymän piirtämisestä ja alustavat kaikki tason objektit törmäyksentunnistusta varten.

Pelaaja luokka:

Pelaaja luokka kuvaa ohjelman ohjattavaa hahmoa, tässä tapauksessa punaista palloa. Pelaaja luokka perii QGraphicsEllipseItemin. Pallon kuvaamisen lisäksi Pelaaja luokassa tapahtuu kaikki törmäyksen tunnistus, pelaajan liikuttaminen ja tasokohtaisten pisteiden päivitys.

Metodi update_pelaaja hoitaa pelaajan liikuttamisen ja törmäyksen tunnistuksen. update_pelaaja metodia kutsutaan jatkuvalla luupilla Scene luokan timerEvent metodissa pelin ollessa käynnissä. Pelaaja luokka toimii siis tiiviissä yhteistyössä Scene ja Taso luokkien kanssa.



Suuntaa antava kuva ohjelman luokkarakenteesta.

Algoritmit:

Ohjelman avautuessa käynnistetään kello, joka tikittää 16ms välein. Tämän kellon tahdissa päivitetään ohjelmaa, pelaajaa ja tarkastellaan, onko törmäyksiä tapahtunut. 16ms välein ohjelma tarkistaa yksittäin useita asioita kuten:

Onko näppäimistöstä painettu nappeja? Jos on, pelaajaa pyritään liikuttamaan.

Pystyykö pelaaja liikkumaan halutun matkan vai tapahtuuko törmäys Maa palaan? Jos törmäys tapahtuisi ei pelaajaa voida liikuttaa. Tämä tarkastus toimii luomalla uusi väliaikainen pelaajaa vastaava objekti ja asettamalla se kohtaan, johon haluttaisiin liikkua. Jos tämä uusi objekti aiheuttaa törmäyksen, ei pelaajaa liikuteta. Jos pelaaja siirrettäisiin ensin ja sitten tarkasteltaisiin, tapahtuiko törmäys, pelaaja tulisi liikkumaan edestakaisin jatkuvasti törmätessään maahan/seiniin.

Törmääkö pelaaja piikkiin, kolikkoon tai maaliin? Jokaisen törmäyksen jälkeen ohjelma reagoi tarkoituksenmukaisella tavalla.

Kellon sijasta voitaisiin käyttää While looppia, mutta mielestäni kellon käyttäminen on paljon käytännöllisempää. Kellon voi helposti pysäyttää jopa luokan ulkopuolelta ja QGraphicsScenellä on valmiina timerEvent metodi, jota voidaan käyttää yhdessä kellon kanssa.

Muut ohjelmassa esiintyvät algoritmit ovat oikeastaan vain for looppeja joilla käydään läpi tason dataa ja käydään läpi pelaajien saamia pisteitä.

Tietorakenteet:

Ohjelman tärkein rakenne on kaksiulotteinen lista, jossa tasojen rakenne säilötään. Kaksiulotteinen lista oli mielestäni hyvä valinta, koska sitä on erittäin helppo käsitellä ja iteroida. Tässä tapauksessa listan pituus on 20 ja jokaisen listan kohdan sisällä on toinen 20 numeroa pitkä lista. Tällä tavalla voidaan muodostaa 20x20 ruudukko. Tämä ruudukko voidaan myös järjestää koodissa neliön muotoiseksi ja tämä auttaa tason rakenteen visualisoinnissa, kun tasoa rakennetaan. Tasoa on myös helppo muokata, riittää että vaihtaa listan numeroita. Toinen vaihtoehto olisi ollut säilöä tason dataa erillisessä tiedostossa esimerkiksi merkkijonona. Jos olisin toteuttanut ohjelmaan graafisen tasonmuokkauseditorin, olisi tason datan säilöminen tiedostossa ollut pakollista tallentamisen takia. Tätä en kuitenkaan päättänyt tehdä ja tästä johtuen tason dataa voi säilyttää itse ohjelmassa, koska pelaajan ei tarvitse päästä muokkaamaan tasoa ajamalla ohjelmaa.

Tiedostot:

Ohjelma lukee ja tallentaa pelaajaan saavuttamat pisteet tekstitiedostoon. Pisteet säilötään tiedostossa omilla riveillään. Pisteitä voi helposti lisätä top scores taulukkoon lisäämällä numeroita scores tiedostoon omille riveilleen. (Huom top scores näyttää vain 5 korkeinta numeroa, jotka tiedostosta löytyvät ja jos samalla rivillä on usea numero, jää koko rivi huomioimatta.) Ohjelma tarkastaa automaattisesti, jos scores tiedoston dataa ei voida muuttaa numeroksi ja ohittaa tällaiset tapaukset. Ohjelmaa voi ajaa myös tyhjällä scores tiedostolla, mutta scores tiedoston on oltava olemassa.

Ohjelmassa käytetään vakiot nimistä python tiedostoa, johon on tallennettu yleisessä käytössä olevia arvoja kuten luotavan ikkunan korkeus ja yhden ruudun leveys. Tämän avulla on helppoa halutessaan muokata näitä arvoja ja niitä ei tarvitse vaihtaa itse koodin jokaisessa kohdassa.

Testaus:

Ohjelma toimintaa testattiin koko projektin ajan aina kun ohjelmaa ajettiin. Törmäyksen tunnistus, liikkuminen ja muut vastaavat toiminnot saatiin testattua parhaiten pelaamalla luotua peliä. Mielestäni kaikki edellä mainitut toiminnot toimivat kiitettävästi.

Tein myös muutaman yksikkötestin, jotka testasivat ohjelman pyörittämistä virheellisellä scores tiedoston datalla ja virheellisellä tason datan sisällöllä ja rakenteella. Ohjelma läpäisee kaikki testit, mutta ongelmana on testit itse. Ainut tilanne jossa ohjelman on tarkoitus heittää poikkeus on se, kun tasoa ei pystytä luomaan tason datan virheellisen rakenteen takia. Koska muissa datan virhetapauksissa on ongelma tarkoitus vain ohittaa (esimerkiksi tason datassa kirjain tulkitaan tyhjäksi kohdaksi) en ollut täysin varma kuinka näitä asioita pitäisi oikeaoppisesti testata.

Ohjelman tunnetut puutteet ja viat:

Scene luokka on liian iso: Scene luokkaan on oikeastaan tungettu aina kaikki uudet ominaisuudet, joita ohjelmaan on lisätty ja tähän olisi varmasti siistimpiäkin tapoja. Tätä voisi ajan kanssa muokata, mutta en halunnut lähteä tekemään suuria muutoksia ohjelman rakenteeseen projektin loppuvaiheessa.

Koodissa on kohtia, joissa on ratkaistu ongelmia huonolla tyylillä: Joissain tapauksissa asiat eivät toimineen täsmälleen ajatellulla tavalla joten koodiin oli lisättävän hieman turhan näköisiä kohtia näiden asioiden ratkaisemiseksi. Esimerkki:

```
self.scene.removeItem(kolikko)
kolikot.remove(kolikko)

# tämä osa täytyi sisällyttää jotta scoren määrä ei olisi riippuvainen kolikon ja pelajaan
# törmäyksen pituudesta

if kolikko not in kolikot:
    self.score += 1  # lisätään scoreen 1
```

Kolikko ei poistunut välittömästi ja scoreen lisättiin useita pisteitä.

Jos tasossa olisi tasainen maa, jossa on yhden ruutu leveä aukko, pystyy pelaaja kulkemaa aukon yli tippumatta. Tällaisia kohtia ei tasoissa tällä hetkellä ole. Ongelma johtuu siitä, että pelaaja ei liiku pikseli kerrallaan ja saattaa ”hypätä kolon yli”. Jos kolon päälle pysähtyy, pelaaja tippuu koloon niin kuin kuuluukin.

Yksikkötestien laatu/toteutus saattaa olla puutteellinen, en ole varma mikä olisi ollut oikea tapa näiden testien toteutukselle.

3 parasta ja 3 heikointa kohtaa:

3 parasta:

Yleinen toimivuus ja sulavuus:

Ohjelma toimii sulavasti ja tarkoituksenmukaisella tavalla.

Törmäyksen tunnistus:

Törmäyksen tunnistus ei ole pikselin tarkka, mutta toimii silti mielestäni tarkasti, eikä pelaaja jää kiinni esimerkiksi seiniin.

Pelin fysiikat ja tasot:

Pallon liike tuntuu luonnolliselta ja tasot ovat toisiinsa nähden melko erilaisia ja mielestäni aika hauskoja.

3 Heikointa:

Pelin ulkonäkö:

Alun perin oli tarkoitus lisätä peliin kunnon grafiikkaa mutta tällä kertaa ohjelman käytännöllisyys meni ulkonäön edelle ajan puutteen takia.

Ominaisuuksien puutuminen:

Ohjelman pitäisi kyllä täyttää vaativan tason kriteerit, mutta tästä huolimatta olisin halunnut lisätä toimintoja ja ominaisuuksia.

Koodi:

Ohjelmakoodi voisi varmasti olla tiiviimpää ja siistimpää.

Poikkeamat suunnitelmasta

Suunnitelmasta poiketen ohjelmasta puuttuu liikkuvat viholliset ja aloitus menu ajan puutteen takia. Muuten ohjelman rakenne on karkeasti saman tyyppinen, kun suunnitelmassa oli mietitty. Törmäyksen tunnistus tapahtui myös eri luokassa kuin alun perin olin ajatellut. Toteutusjärjestys oli täsmälleen sama kuin suunnitelmassa.

Toteutunut työjärjestys ja aikataulu

Työjärjestys karkeasti: Ikkuna, scenen alku, Taso, Maa, Pelaaja, Piikki, Maali, Kolikko, lisää tasoja, voitto näkymä, pistelaskuri ja topscores, pikanäppäimet. Projektin aloittaminen viivästyi huhtikuun alkuun, mutta siitä lähtien projekti edistyi melko tasaisesti.

Arvio lopputuloksesta

Yleisellä tasolla olen erittäin tyytyväinen ohjelman laatuun ja toimivuuteen. Ohjelma ei ole täydellinen mutta toimii tarkoitetulla tavalla ja täyttää kriteerit. Peli toimii sulavasti ja törmäyksen tunnistus on hyvä. Peliä on helppo pelata uudestaan ja täysien pisteiden saaminen pelistä on haastavaa.

Ohjelmassa voisi olla enemmänkin ominaisuuksia ja sen ulkoasu voisi olla nätimpi.

Mielestäni luokkajako ja tietorakenteet on valittu hyvin ja en niitä hirveästi lähtisi muuttamaan. Ohjelmaa on erittäin helppo laajentaa esimerkiksi luomalla uusia tasoja. Riittää että luo samanlaisen kaksiulotteisen listan, johon tekee oman tasonsa ja lisää tämän listan osaksi taso_data listaa, joka säilyttää kaikki tasot. Myös esimerkiksi peli ikkunan kokoa voi muuttaa vaihtamalla ruudun leveyttä ja korkeutta vakiot tiedostosta.

Viitteet

Qt5 dokumentaatio

Youtube kanavat Codin With Russ, Tech With Tim ja Tanny Chung

Satunnaisissa ongelmissa stackoverflow